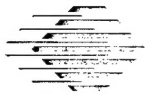
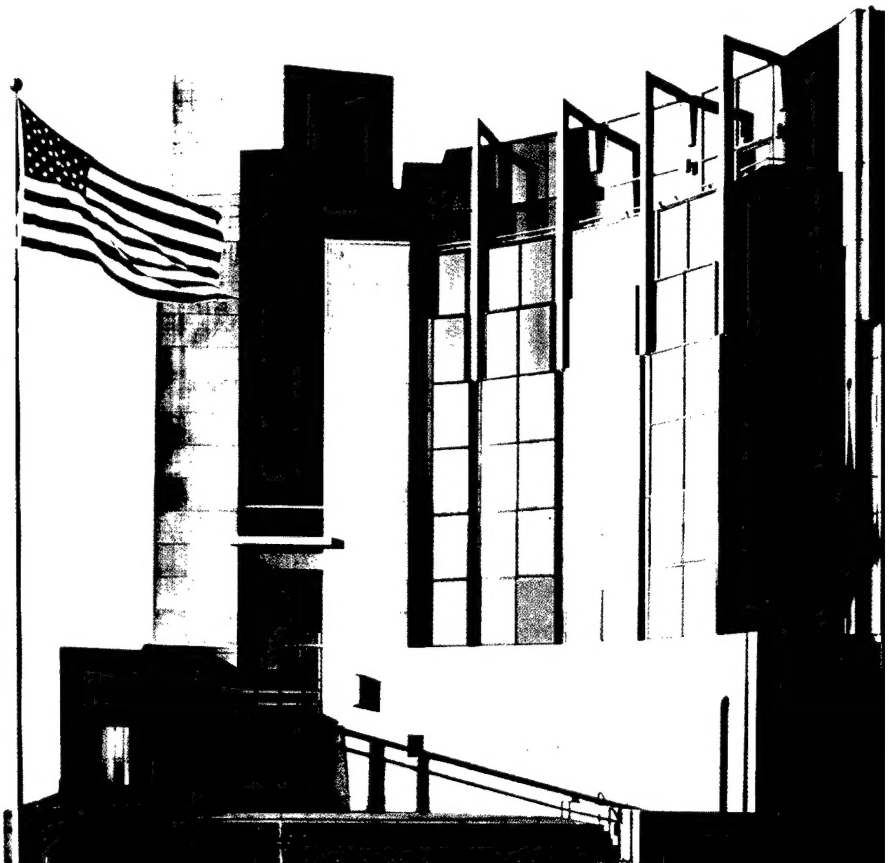


Carnegie Mellon
Software Engineering Institute

Model-Based Verification: A Technology for Dependable System Upgrade

David P. Gluch
Charles B. Weinstock

September 1998



TECHNICAL REPORT
CMU/SEI-98-TR-009
ESC-TR-98-009

Carnegie Mellon University does not discriminate and Carnegie Mellon University is required not to discriminate in admission, employment, or administration of its programs or activities on the basis of race, color, national origin, sex or handicap in violation of Title VI of the Civil Rights Act of 1964, Title IX of the Educational Amendments of 1972 and Section 504 of the Rehabilitation Act of 1973 or other federal, state, or local laws or executive orders.

In addition, Carnegie Mellon University does not discriminate in admission, employment or administration of its programs on the basis of religion, creed, ancestry, belief, age, veteran status, sexual orientation or in violation of federal, state, or local laws or executive orders. However, in the judgment of the Carnegie Mellon Human Relations Commission, the Department of Defense policy of "Don't ask, don't tell, don't pursue" excludes openly gay, lesbian and bisexual students from receiving ROTC scholarships or serving in the military. Nevertheless, all ROTC classes at Carnegie Mellon University are available to all students.

Inquiries concerning application of these statements should be directed to the Provost, Carnegie Mellon University, 5000 Forbes Avenue, Pittsburgh, PA 15213, telephone (412) 268-6684 or the Vice President for Enrollment, Carnegie Mellon University, 5000 Forbes Avenue, Pittsburgh, PA 15213, telephone (412) 268-2056.

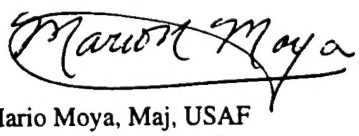
Obtain general information about Carnegie Mellon University by calling (412) 268-2000.

This report was prepared for the

SEI Joint Program Office
HQ ESC/AXS
5 Eglin Street
Hanscom AFB, MA 01731-2116

The ideas and findings in this report should not be construed as an official DoD position. It is published in the interest of scientific and technical information exchange.

FOR THE COMMANDER



Mario Moya, Maj, USAF
SEI Joint Program Office

This work is sponsored by the U.S. Department of Defense.

This work was created in the performance of Federal Government Contract Number F19628-95-C-0003 with Carnegie Mellon University for the operation of the Software Engineering Institute, a federally funded research and development center. The Government of the United States has a royalty-free government-purpose license to use, duplicate, or disclose the work, in whole or in part and in any manner, and to have or permit others to do so, for government purposes pursuant to the copyright license under the clause at 52.227-7013.

This document is available through Asset Source for Software Engineering Technology (ASSET): 1350 Earl L. Core Road; PO Box 3305; Morgantown, West Virginia 26505 / Phone: (304) 284-9000 or toll-free in the U.S. 1-800-547-8306 / FAX: (304) 284-9001 World Wide Web: http://www.asset.com / e-mail: sei@asset.com

Copies of this document are available through the National Technical Information Service (NTIS). For information on ordering, please contact NTIS directly: National Technical Information Service, U.S. Department of Commerce, Springfield, VA 22161. Phone: (703) 487-4600.

This document is also available through the Defense Technical Information Center (DTIC). DTIC provides access to and transfer of scientific and technical information for DoD personnel, DoD contractors and potential contractors, and other U.S. Government agency personnel and their contractors. To obtain a copy, please contact DTIC directly: Defense Technical Information Center / Attn: BRR / 8725 John J. Kingman Road / Suite 0944 / Ft. Belvoir, VA 22060-6218 / Phone: (703) 767-8274 or toll-free in the U.S.: 1-800 225-3842.

Table of Contents

List of Figures

Abstract

This is a preliminary report on the foundations of model-based verification for engineering software system upgrades. It describes the historical background and technical foundations for the approach and begins to provide a basis for the transition of model-based verification into practice. Critical technical and procedural issues that have been or are being addressed to ensure successful transition are examined. The report is aimed at providing technical insight and understanding for software management and engineering personnel on this emerging technology for verification of software system upgrades.

Acknowledgements

The authors would like to thank the technical reviewers of this work, Peter Feiler, Patrick Place, and William Wood, for their insightful comments and recommendations. We would also like to thank Jared Brockway for his contributions to the section on the SPIN model checker and review comments on the report. Douglas Mosurak and William McSteen helped us to prepare the final manuscript, and their help is gratefully acknowledged. Finally, we extend our thanks to the members of the technical staff within the Dependable System Upgrade initiative for stimulating discussions on this topic. Their perspectives helped us to focus and clarify the technical foundations of model-based verifications.

Executive Summary

Work is underway at the SEI to evaluate and codify innovative approaches to reduce software errors and increase the quality of upgrades to mission-critical software systems. One of the key technologies of these approaches is model-based verification. Model-based verification technologies are built upon the results of an increasing body of research, anecdotal evidence, promising software pilot studies, and successful industrial applications of model-based analysis in digital hardware design and protocol analysis.

In the upgrade of a software system, model-based verification relies on models and higher design representations as the foundation for verification and test activities. The action of generating models early in the upgrade process establishes a systematic software verification practice for understanding an upgrade and identifying errors. The results of these error detection activities are used as the basis for testing by defining test strategies, test cases, and critical areas that require focused or more extensive testing. A high-level overview of these activities is shown in Figure 1.

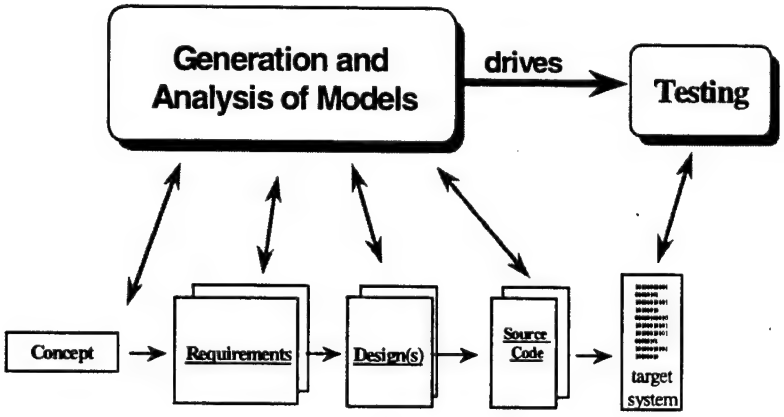


Figure 1: Model-Based Verification Activities

The central foundation of model-based verification is a selective and *pragmatic* application of formal methods in the form of simplified models that focuses on error (defect) identification rather than formalized specification or proofs. By judiciously choosing when, to what detail, and to what extent to employ them, the model-based verification approach capitalizes on the advantages provided by formal methodologies without incurring the costly overhead normally associated with them.

A promising basis for the use of model-based verification in software is found in its success in the form of model checking for digital hardware and complex protocol systems. Some representative examples of the published successes of model checking include

- identifying a critical error in a complex protocol chip that had undergone conventional reviews, simulation, and testing and was ready for production
- detecting an error in a hardware protocol standard (IEEE standard Futurebus+) that had previously been extensively discussed, simulated, and implemented
- providing guidance in correcting and uncovering the causes of a complicated error in a microprocessor. The error was detected during operational computer configuration testing, when it is difficult to ascertain underlying causes using traditional analysis techniques. This work also demonstrated that with model checking the error could have been detected earlier in the development effort, before the design was implemented in silicon.

Based on the success of model-based verification techniques in other applications, there are numerous potential benefits of these approaches for improving the dependability of software upgrades. Two specific benefits are

- flexibility. The methods can be applied to all or only the most critical portions of the system. Portions of the system can be modeled at differing levels of abstraction as determined by need.
- effectiveness. The methods are useful for detecting both basic and more subtle errors (and hence potentially more costly) errors, many of which are not identified even in extensive testing.

Model-based verification integrates established software modeling techniques, formal methods, and innovative model checking approaches into a systematic engineering verification and testing practice. Although this practice is useful at all phases of system development, this report concentrates on its use for engineering dependable software upgrades. Specifically, these efforts address the technical, process, and transition issues relating to model-based verification in software engineering upgrade practices.

1 Background and Introduction

Rapid, perpetual, and complex change characterizes today's DoD software-dependent mission critical systems. The Department of the Air Force's *Guidelines for Successful Acquisition and Management of Software-Intensive Systems: Weapon Systems, Command and Control Systems, Management Information Systems* is quite definitive in characterizing the importance of changes made to DoD software systems. With statements like "the *Software Crisis* has really been the *Maintenance Crisis*" and "... the typical cost to maintain a software product is from *60% to 80% of total life cycle costs*" [USAF 96], the problems experienced with maintaining these systems and the high cost of the latest technology appear to be intertwined.

Further complicating this situation of ubiquitous change to software systems is the error prone character of those changes. Studies show that the probability of making an error, depending upon the level of complexity of the change, ranges from 50 to 80% [Hetzel 88, Ostrand 88]. The costs associated with errors in software can amount to as much as 70% of the cost of developing that software [Harel 87]. These data point to the need to address error reduction as a critical issue in software-intensive upgrades.

In meeting the challenges to reduce software upgrade errors and increase the quality of mission-critical software systems, the SEI is codifying a set of software engineering techniques and practices for verification and test, known as model-based verification (MBV). MBV can be used to identify and correct errors in the requirements, design, or implementation of a software-dependent system upgrade.

At this point in the evolution of MBV practices, the technical focus is on techniques, principally formal approaches, that address the idiosyncrasies and problems associated with digital systems. In particular, techniques that have shown promise for software and have been successfully employed in research or commercial endeavors (e.g., model-checking technologies in digital hardware verification) are being investigated.

1.1 Goals of Model-Based Verification Efforts

A principal role of the SEI is to investigate the efficacy and appropriateness of promising engineering techniques and to unite them into a set of software engineering practices. Substantial leverage is found in capitalizing on the results of a wide variety of research and engineering studies and incorporating those results into a framework appropriate to improve software engineering practice. Earlier work in formal methods at the SEI addressed their use for reactive systems, investigated the state of industry applications of formal methods, discussed their use in

safe-critical software and in describing architectures, and defined a classification scheme for formal specification techniques [Clements 95, Place 89, Place 90, Place 93].

The current work in model-based verification is building upon an increasing foundation of promising techniques, especially formal techniques, and adapting the underlying principles and methodologies to improve software system upgrades. In the broader context, model-based verification is seen as an integral technique in verification and testing practices that reduce the occurrence of errors in software upgrades.

1.2 Structure and Content of the Report

This report discusses the technical foundations of model-based verification by focusing on generalized modeling techniques and formal methods that address the discrete nature of software systems. The goal of this work is to provide technologies and practices that improve the dependability of software upgrades. Since this is an active area of investigation within the SEI, the technologies presented here will evolve and expand. Consequently, rather than a definitive statement on the technology, this report is a preliminary presentation on the nature and potential of model-based verification for software system upgrades. Subsequent technical reports and publications will describe the evolution of technology and practices issues associated with the transition of model-based verification approaches for software system upgrades.

Section 2 is an overview of model-based verification. The technical aspects of model-based verification are discussed in Section 3. Model-checking technology and its uses are presented in Section 4. Section 5 summaries this report and discusses future directions.

2 Model-Based Verification

Model-based verification (MBV) is an engineering practice for identifying and correcting errors. It can be applied in the development as well as in the upgrade of complex software systems. The technical foundation of the approach involves the selective use of varying levels of abstraction and formalism in the systematic generation and analysis of "essential" models of a system. Essential models are simplified (abstract) representations that capture the essence of a system, rather than provide an exhaustively detailed description.

It is the reduced complexity afforded by essential models that enables the judicious incorporation of the rigors of, while minimizing the adverse, high-cost impacts associated with formal methods. Through the selection of only critical parts of the system and/or abstracted perspectives of the entire system, a reviewer using model-based techniques can apply the appropriate method and tailor the model to the idiosyncrasies of an individual system.

Model-based verification benefits from and incorporates the work of numerous researchers and practitioners in a number of diverse disciplines. The practices and principles of model-based verification are an amalgamation of a variety of different modeling and analysis techniques already established in software engineering practices, used in other engineering domains (e.g., commercial microprocessor design), or currently emerging in academic and corporate research communities [Clarke 96b, Easterbrook 97, Heitmeyer 96]. In particular, the ideas and results of these reports have helped to formulate and evolve the foundations of model-based verification approaches. Because most of these techniques involve the selective, focused use of formal methods (e.g., within MBV in generating essential models) these techniques have been termed *lightweight* formal methods [Jackson 96a, Jackson 96b].

2.1 The Model-Based Verification Paradigm

Verification is used here in the broad sense of an engineering practice that evaluates the artifacts of software development and maintenance. This is to determine that these artifacts (specifications) agree with their antecedents and ultimately with the desired system. Their relationships are represented in Figure 2.

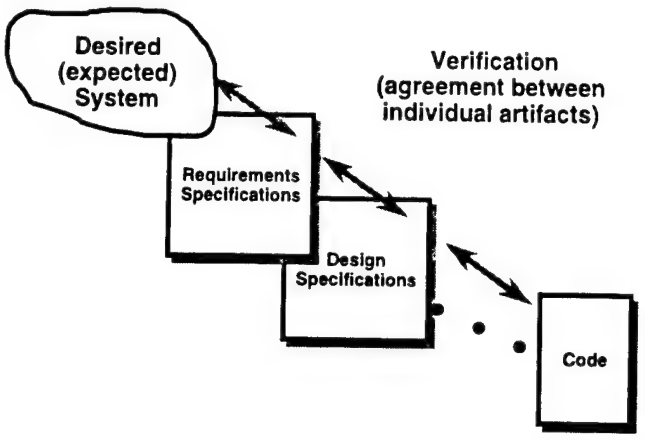


Figure 2: Verification and Validation

The selective application of formal methodologies in the form of models, and the focus on error identification and correction, rather than proof-of-correctness, distinguishes the model-based verification paradigm from more traditional formal methods approaches. When formal methods are used for software engineering, both the "artifact being verified" and its antecedent must be in a formal representation. Verification on an artifact therefore involves proving a correspondence between the statements in the artifact itself and those in its antecedent. These proofs are based on the explicit and precise delineation of assumptions and the mathematical reasoning rules that characterize formal methods.

In contrast to approaches where formal methods pervade the specification and verification processes, for model-based verification, only the essential models need be in a formal representation. These formalized representations, essential models, are less detailed (simplified) portrayals of the artifact to be verified, are distinct from other engineering artifacts, and are analyzable. Figure 3 shows the relationships among the various artifacts. The essential models are direct representations of the "artifact to be verified." The antecedent artifact and the desired system (as described by requirements or other specifications) are used to define desired properties. In addition to the identification of errors that occurs during the generation of the models, the models themselves can be analyzed (checked) to determine if they possess the desired properties.

The number, level of formalism, and scope of the models used are adjusted to meet the particular objectives of the verification activity and the complexity of the system. In some cases only a handful of models or only models of minimal formalism are required. In others, more rigorous formal models are developed that can be formally analyzed using automated analysis tools. The range of formalism can extend from using a basic state model to employing a complete axiomatic formal system with accompanying syntax, semantics, and inference rules.

Both the process of modeling and the process of checking (whether manual or automated) uncover errors in the "artifact to be verified." In generating the model(s) of the artifact, formalism provides a systematic framework and procedure. The discipline and logic inherent in the modeling activity uncovers numerous omissions, inconsistencies, and logic errors. This has often been cited as one of the major benefits of using formal methods in the specification process [NASA 95, Rushby 93]. Thus, simply modeling, even developing a highly abstracted model of only part of a system, can provide substantial insight and identify significant errors [Clarke 95, Dill 92, Raimi 97]. The checking of models (especially automated checking) can identify more complex errors, often those that defy detection by conventional inspection processes, simulations, and testing [Clarke 96a].

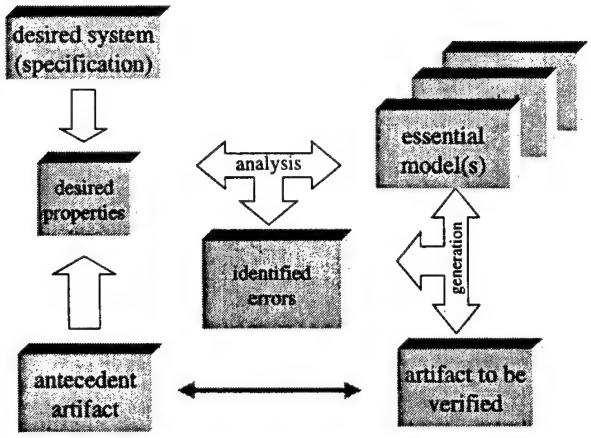


Figure 3. Essential Models in Model-Based Verification

The engineering decisions on the level of detail of essential models, their perspectives, and their scope are key in the application of model-based verification. These decisions are part of model-based verification practice and are based upon the specific needs of the individual development or upgrade effort. The levels of formalism used for a software product may vary between upgrades throughout the useful lifetime of that product.

In addition to models founded upon formal methods, the models that are employed within model-based verification encompass a variety of mathematical-based techniques that are not classified as formal methods. Generally, the terminology of formal methods denotes discrete mathematical and predominantly axiomatic techniques that address the problematic issues that emerge within digital systems [Rushby 93]. Within model-based verification practices, semi-formal modeling techniques and other modeling approaches that address the diverse and potentially problematic technical aspects of complex digital systems can be employed. For example, the use of Generalized Rate Monotonic Analysis [Klein 93] models may be used as part of model-based verification practices.

2.2 Context and Related Work

As a software engineering practice, model-based verification leverages the precision, discipline, and thoroughness afforded by formal methods to address the problems that have plagued many software upgrades. There have been others who have advocated the integration of formal methods into software engineering practice [Meyer 85, Rushby 93]. Generally, these efforts have focused on formal methods as a basis for both specification and verification [Parnas 98, Rushby 93, Vienneau 93].

Different perspectives on integrating formal methods into software engineering practice have been presented by other researchers. Specifically, Michael Jackson points out that "...although the claimed advantages of formal methods are correctness and reliability, the methods are implicitly presented as constructive development methods—that is, methods for describing new problems, and inventing and designing novel solution structures with which correctness and reliability can be assured." He also notes that formalisms offer potential as "sharply focused micro-methods, supporting specialized small-scale tasks of analysis and detailed design." [Jackson 98]. Parnas feels that the concepts underlying formal methods should be an integral part of routine software engineering practice and not a separate "add-on" [Parnas 98].

Researchers employ formal modeling and checking as tools for verification or validation of software, regardless of the specification language employed in the development [Atlee 93, Clarke 96b, Easterbrook 97, Miller 98, Wing 96]. In particular, the ideas discussed here are directly related to the application of formal methods and tools as "spot checks" to test for errors in Independent Verification and Validation (IV&V) of software requirements for safety-critical systems as explored in Easterbrook [Easterbrook 98]. Other researchers address adoption issues by developing presentation styles for formal methods that are more familiar to domain engineers (e.g., tabular representation of state machines) [Heitmeyer 96, Leveson 94, Parnas 92].

In the broad context of the use of formal methods in software, model-based verification can been seen as incorporating the ideas of partial and focused formalisms, providing much of the value afforded by formal methods while minimizing the pitfalls accompanying their adoption. This lightweight application [Jackson 96b] of formal methods in the form of partial abstract models contrasts with the use of formal methods as detailed and rigorous specification techniques. In formal specification the objective is to create specifications amenable to direct formal analysis. Model-based verification typifies a lightweight approach to formalism that focuses on simplified models that can be analyzed and capture the essence of the system.

2.3 Model-Based Verification in Software Upgrades

While model-based verification can be applied in the development of a new software system, the focus of work at the SEI is to apply emerging model-based techniques and related support tools to improve the quality of software upgrades.

Model-based verification is especially useful at higher levels of representation (e.g., requirements capture, system design, requirements and design analysis). This focus on verifying requirements and design representations using model-based verification techniques, in contrast to relying principally on testing for identifying errors, can help ensure a more reliable and predictable upgrade. Specifically, model-based verification techniques can be used to augment existing review practices and can help to focus regression testing of upgraded code.

The partial analysis capability of model-based approaches enables a software engineer to focus on the areas of change without modeling the entire system. In addition, through simple modifications to entire system models, created in earlier design and verification efforts, changes in a particular part can quickly be reflected in analyses of the total system. These analyses of updated models provide insights into the global impacts of changes and can form the basis for an evolutionary upgrade practice, where models are central to the development and maintenance of a software system.

In addition to detecting errors in an upgrade, model-based verification practices and the models that are created can form the basis for proactive upgrade design activities. In proactive design, models can be employed to assess the impact as well as the extent of potential changes prior to their implementation, identifying both desired and adverse consequences. These model-based practices can be integrated into broad design approaches for upgrade.

This basic model-based verification paradigm can be integrated into existing software engineering processes, or form the basis for establishing a distinct set of model-based verification processes within an upgrade effort. The substantial potential for the use of models, as well as model-based verification, as a basis for improving the dependability (quality) and affordability of software upgrades, is a principal area of investigation within the dependable system upgrade initiative at the SEI. This and subsequent SEI reports will focus on the use of model-based verification techniques in engineering of dependable software upgrade.

3 Technical Aspects of Model-Based Verification

Software development is an intellectual endeavor that involves the description of what a system is and how it behaves, both internally and in the context of its environment. This point is made very clearly in the observation that "to develop software is to build a *Machine*, simply by describing it" [Jackson 95]. Given this perspective, descriptions and the languages that are used for them are central to successful software engineering.

The descriptions of desired systems or updates to them are embodied in the artifacts, the "documents," of software development. The effort of producing software can be seen as a progression in the description of a system, extending from higher level descriptions (requirements) that primarily involve a language of the system's environment (domain), to descriptions in the idiosyncratic programming languages of "software." Thus, when a desired system is correctly described in a programming language and executed on the target machine, it is a product that "possesses" desired characteristics and behavior within its environment.

It is important to recognize that this evolution in the development of a software product is twofold. One is the progression of the context of the description, from application domain to internal computer and software domains. A second is a change in the nature of the description itself. This change can be viewed as a transition from static prose, often expressed in natural language, to a dynamic "embodiment" of the system in the form of executable code.

Code and some other intermediate descriptions used in software engineering (e.g., simulations) are more than static representations of structure or behavior. While varying in the level of detail, these descriptions and the language employed for them share common characteristics of the reality they portray, often mirroring the system's behavior. In an engineering sense, many of these descriptions are analogs (models) of the real system they describe [Jackson 95].

3.1 Models

Models have been defined in a variety of ways. The meaning and use of the term model depends on the application domain. Even when employing similar mathematical techniques, the models used by a geneticist are very different from those used by an aerospace engineer. This dilemma, arising out of the context of use of the term model, is further complicated by the fact that different kinds of models can be identified. For example, Ackoff [Ackoff 62] defines three kinds of models:

- *iconic* (e.g., a model airplane, an icon of reality). This is a model that is a physical, perhaps scaled down and simplified, representation of the reality. It is generally more than just a picture. In an iconic model, some details may have been eliminated but the key structural aspects of the reality are captured visually. [Included here are distinct "visual" representations of the system that use electromagnetic perspectives that are not visible to human beings (e.g., an infrared model of an object).]
- *analogic* (e.g., analog of reality). This is a model of a system that mirrors principal behavioral characteristics of a system within another domain. For example, the values of the voltage and current in an electrical circuit can be used to model the pressure and flow of a liquid.
- *analytic* (e.g., differential equations of motion). These are mathematical statements where the variables' model physical parameters in the reality being modeled and the operations and other mathematical properties reflect the behavior of the reality. By analyzing the mathematical system one can gain insight into the properties of the real system.

The type of models that are used in model-based verification involve primarily the last two of these: analogic and analytic. The "definition" that we will adopt here captures the inherently descriptive nature of software and reflects a number of perspectives including:

- "...an abstraction of something for the purpose of understanding it before building it." [Rumbaugh 91]
- "An abstraction of reality that [itself] shares important behaviors, properties, and structure with that reality." [Jackson 95]

It is the ability to both provide insight into a system (understanding) and exhibit shared properties with that system that characterize the "models" used in model-based verification.

3.2 Formalism in Modeling

Most of the modeling approaches employed in model-based verification are based upon some level of formalism and involve the application of a formal methodology[1]. Vienneau [93] adapts the definition provided in Wing [90] to define a formal method in software development as "a method that provides a formal language for describing a software artifact (e.g., specifications, designs, code) such that formal proofs are possible, in principle, about properties of the artifact so expressed."

[1] There are a number of comprehensive discussions on formal methodologies in the literature including [NASA 95, NASA 97, Vinneau 93, Wing 90, NIST I]. These provide both non-technical and theoretical descriptions of formal methods.

For the purposes of this report, formalism, specifically a formal modeling approach, is viewed as a collection of principles and practices that are built upon

- well-defined language of expression and inference (i.e. a set of symbols and clearly defined rules for their use, generally mathematical or logic rules)
- meaning assigned to the symbols of the language (i.e. a set of procedures for giving meaning to the symbols and interpreting the rules of use of the language in terms of these meaningful symbols)

Within model-based verification, the formalism of software engineering models is used to capture the essential properties and behavior of a system. A formal modeling system has a well-defined language and methodology that includes mathematical and logical concepts and constructs that enable formal deduction (reasoning and inference about a system) and "proof" of properties (theorems) about that system.

3.3 Formal Methods for Verification

While formal methods have long been advocated for software engineering, they have not been widely used. Formal methods have been applied in cleanroom engineering [Linger 94, Mills 87] and other projects. Sixty commercial projects were identified and twelve reported on in detail in an international survey [Craigen 93a, Craigen 93b]. The Formal Methods in Europe (FME) series of symposia has focused on issues relating to the industrial use of formal methods [Gaudel 96]. However, the results of the adoption of formal methods have been mixed [Pfleeger 97] and there is controversy over their efficacy for industrial applications [Computer 96]. The more notable applications of formal methods have been in safety-critical and high assurance systems [Clarke 96b, Jackson 98, NASA 95, NASA 97, Vienneau 93].

A frequently advocated implementation of formal methods for software development involves their use in the specification, design, and verification of a system [Parnas 98]. In this case, formal verification entails complicated and generally quite costly formal proofs or proof checking (sometimes supported by tools for theorem proving) to establish correctness of the specification.

3.4 Formal Methods and Essential Models

In contrast to a full specification and verification of a system by formal languages and associated proof systems throughout the software engineering effort, model-based verification involves the selective application and focused use of formal methods. Specifically, formal methods are employed for detecting and correcting errors rather than developing detailed specifications or proving correctness.

In model-based verification, well-formed "essential" models are created using formalism, and analyzed relative to expected behavior and properties. Generating essential models involves creating "simplified" models that accurately portray the important properties of a system

while disregarding (abstracting away) the less important. This abstraction and modeling process involves engineering choices in four distinct dimensions:

- formalism: what formalism to apply?
- abstraction: what level of detail is needed?
- perspective: what modeling views of the system are required?
- scope: what parts of the system need to be modeled?

From a practical perspective, these issues involve decisions on what modeling approaches to employ and what aspects of the system should be modeled, i.e., identifying what is important about the system in its use.

The context, the nature of the problem domain, is particularly influential in these choices and the decision process is one of matching between domain and model, rather than simply selecting. There is a need to match the specific characteristics of the environment with the particular characteristics of the modeling approach. These dimensions are not mutually exclusive but there is often interplay among them. For example, deciding scope and perspective may involve a trade-off between modeling the whole system from a single perspective or critical parts of the system from multiple perspectives and at various levels of abstraction.

3.4.1 Formalism

What is the appropriate level of formalism to employ for this system? This decision often is a programmatic as well as a technical one. While the extensive use of formalism is appropriate for high assurance and safety critical systems, the cost-effective application of formalism in most system development and upgrade efforts entails an impact assessment of a complex interplay of cost and criticality. Formalism is generally very costly and the potential benefits of formalism must be assessed relative to the risks and criticality of a system.

3.4.2 Abstraction

What level of detail is required to achieve the objectives of the analysis? Abstraction is the process of focusing principally on the important details and ignoring (abstracting away) those details which are less important to the problem at hand. If a modeling effort requires a complete "re-specification" of the system, it would likely be overly burdensome to a develop effort.

A promising aspect of model-based verification is evidenced in recent model-checking investigations. These studies suggest that, by using relatively simple, highly abstracted models, significant errors can be successfully identified in complex digital systems [Clarke 95, Raimi 97, others]. Deciding what is important, not only in terms of abstraction level, but also associated scope and perspective, are significant aspects of the engineering decisions involved in using model-based verification.

3.4.3 Perspective

What modeling view or views of the system are required? This addresses the abstract perspective used to represent the system. A perspective could be from that of a user or abstractions associated with a specific feature, function, or capability of a system. For example, in looking at a fly-by-wire computer system for an aircraft, one perspective would be to characterize (model) the system from a fault response perspective. This might involve a model that describes states of the system in terms of the number or types of faults that may occur (e.g., the no fault state, single fault state, loss of roll attitude information). An alternative perspective might be the flight modes of the system (e.g., takeoff, climb out, cruise, landing).

3.4.4 Scope

The scope of the model defines how much (what portion) of a system is to be modeled. In some cases, it is important to model all of the system at a high level of abstraction considering one perspective (e.g., model and analyze the fault response of the entire system). In others, it is necessary to model a specific part of the system in greater detail (e.g., model and analyze the network communication modules). These choices are largely driven by determinations of the critical aspects of the system and its development or upgrade, including both programmatic and technical issues.

A model-based verification effort will involve a set of models of various scopes, perspectives, and levels of abstractions. Establishing guidelines and principles for these decisions is one of the major areas of investigation of the model-based verification work. Specifically, risk-driven and high leverage approaches for making these choices are being investigated.

3.5 Model-Based Verification Techniques

The set of techniques that can be used in model-based verification is broad. As work continues in investigating these techniques, the scope of potential modeling approaches will expand. The range for these modeling techniques includes semi-formal modeling techniques, formal methods, mathematical-based techniques that are not classified as formal methods, and other types of models that address the diverse and potentially problematic technical aspects of complex systems. These other models include, for example, Generalized Rate Monotonic Analysis [Klein 93] models. The semi-formal models include object models (e.g., UML) and those founded in structured design approaches.

At this point in the evolution of model-based verification technology and practice, the technical focus is on state machine modeling approaches. These techniques address the idiosyncratic and problematic issues associated with digital systems. The general set of techniques for potential application in model-based verification include

- state machine models
- sets and relations models

- state exploration approaches
- temporal logic models
- concurrent processes models

In particular, specific techniques that have shown promise for software or have been successfully employed in other disciplines and can be adapted for analyzing software (e.g., model-checking technologies) will be investigated.

4 Model Checking

Model checking is a verification technique that focuses on the automated analysis of finite models of a system. As a technique for analyzing complex state machines, it has been gaining acceptance and producing successful results in digital hardware and protocol analysis [Clarke 96b].

The success of model checking in domains outside of software and its potential for handling complexity in software systems prompted research into its use in software verification. The results of this research and the increasing number of successes of the technique for digital hardware systems provided the impetus and technical foundation for the model-based verification approach. In this context, model checking is viewed as an example of using an automated tool to support the analysis activities within the model-based verification paradigm.

This section presents representative model checking approaches that can be employed in model-based verification practices, by focusing primarily on formal models that are amenable to automated model checking.

4.1 State Machines in Model Checking

Model checking is a form of model-based verification where an automated tool is used to check a system model, generally a state machine model, against its expected properties. The output of the checking tool is either a confirmation that each property is maintained or a non-confirmation—in which case a counter example is provided. For example, a model of a missile launch system is checked against the expected property that the missile will not be launched (engines ignited) unless all the release latches are disengaged. If this property is true for the model, the model checker will confirm its validity. If it is not true, it will show what states or portions of the model (a counter example) will result in the premature launch. Figure 4 depicts the model-checking process where an essential state machine model of a system is "checked" against the system's expected properties.

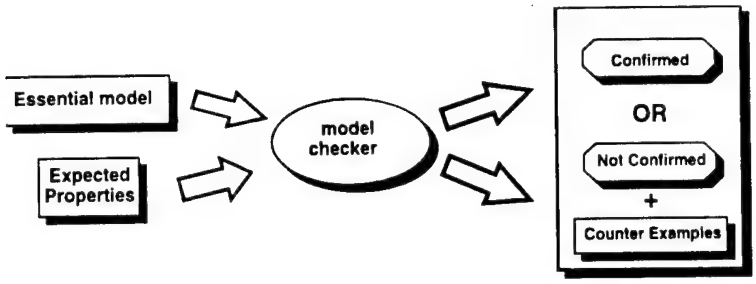


Figure 4: Model Checking

Checking a state machine model involves "exploring" the model's state space to determine the validity of desired properties (claims) about the system. Two distinct approaches for model checking can be identified [Clarke 96b]:

- temporal logic based model checking; this involves a finite state transition model of the system with a temporal logic representation of the expected properties of a system.
- automaton based model checking; this involves an automaton model for both the system itself and its expected properties.

In either of these approaches the checking is readily automated. The challenge in each is to find optimization algorithms and enhanced processing techniques that enable systems of realistic size to be checked (e.g., 10^{1300} states [Clarke 94]).

4.2 Model-Checking Successes

Model-checking techniques and their associated support tools have been successfully employed in a variety of digital hardware and protocol analyses and recently have been used to analyze software systems. This section summarizes some of the successes that have been documented in published reports.[2]

4.2.1 IEEE Futurebus+

Edmund Clarke and his colleagues applied model checking to the Futurebus+ cache coherence protocol [Clarke 95]. In the process of formalizing and verifying the protocol they discovered a number of errors and ambiguities. This is the first time, in their belief, that formal methods have been used to find nontrivial errors in a proposed standard. To do this they used the Symbolic Model Verifier (SMV), a tool that checks if finite-state systems satisfy specifications given in Computational Tree Logic (CTL).

Futurebus+ is described both in an informal language version and by precise attributes. The behavior of an individual cache or memory is given in terms of roughly 300 attributes, so

[2] There have been numerous anecdotal accounts of notable model checking successes that have been circulating throughout the research and development community. These have ranged from detection of subtle errors in fielded email systems to potentially costly errors in high performance microprocessor designs.

they elected to use the informal English version to construct their model, and only to turn to the attributes when necessary to resolve an ambiguity in the English version.

While trying to verify the model of the protocol they found a number of errors, some of which resulted in incoherent caches, and others which resulted in deadlock. They checked configurations of up to three buses and eight processors, with reachable states of up to 10^{30}. This resulted in models of up to 150,000 nodes, a reasonable size which allowed the verifier to give feedback in a matter of minutes. This made it possible to find bugs and try possible fixes very quickly.

What is especially noteworthy about this study is that the specification had been through more than four years of standards reviews by technical experts. Yet despite these efforts, which were based upon less formal review processes, the errors had defied discovery until model-checking techniques revealed them.

4.2.2 Protocol Verification

In 1992 the Murphi protocol description language and verifier were used to verify two hardware designs intended to become commercial products: a directory-based cache coherence protocol and a synchronous link-level communication protocol [Dill 92].

Directory-based cache coherence is a way of implementing a shared-memory abstraction on top of a message-passing network. When using messages that may not arrive in the same order they were sent, the effort to maintain cache coherence in the face of multiple transactions can be quite complicated.

Instead of specifying the complete protocol, the verification team specified a set of properties that were obviously necessary conditions for correct operation. They tried this for varying sized systems (e.g., differing number of main memories and directories, number of caches and processors) Almost all of the errors found were found with a rather small system consisting of one main memory/directory, two processors/caches, and one memory location with one possible value. Hundreds of thousands of states were examined.

Murphi was also applied to the problem of verifying a link-level communications protocol. Verification caught several fundamental errors in the initial design. Redesigning the protocol to be correct and also meet given performance goals was quite difficult. This effort required larger state spaces than the cache coherence protocol above.

As a result of this experience, Dill and his colleagues conclude that automatic formal protocol verification can be a valuable design aid. However, the group can only ensure its effectiveness if the method is used early in the design process, is regarded as a debugging tool, is applied to a high-level abstraction of the system, and the system description is downscaled.

4.2.3 Chip Set Design Checking

AT&T Design Automation decided to commercialize some formal verification technology [De Palma 96]. One part of this technology is FormalCheck, a model checker that was released in 1996. Alpha versions of FormalCheck were used on both internal and external projects. The design of a multimedia chip set was one of these projects. When FormalCheck was ready for testing, the chip design was already believed to be correct and ready for silicon. There was some worry that the results of the evaluation of the effectiveness of FormalCheck would be inconclusive because of this.

To allay these concerns and provide a real test, a version of the design with a single known bug was provided. As expected, most of the checks passed with no errors but several failed. One of the errors found was a potential loss of data due to a write to a full buffer by the controller. This was the known bug. However, FormalCheck identified two other errors, previously unknown to the designers. One could cause the memory access controller to totally ignore a request. The other was a bus arbitration problem, which made it possible for low-priority requests to be blocked indefinitely. These previously known and unknown design flaws were found with only 20 queries to the FormalCheck model checker.

4.2.4 PowerPC 620 Microprocessor

Model checking was employed to diagnose the cause of a design error found during hardware testing of the PowerPC 620 microprocessor [Raimi 97]. The model-checking software used in the model-checking process (Verdict) was developed within Motorola.

During the testing, the error was first detected as a livelock condition while booting an operating system. Subsequent analysis using model checking revealed that the error was the result of a complicated interplay among requests for services, bus snooping, and associated bus access and control actions in the store buffer controller. The model created to analyze this error was relatively small compared to the capability of the tool. The resulting model involved more than two million states. But this level of complexity is beyond that which can be understood in detail and analyzed by an individual engineer without automated support tools.

The design team identified and corrected the error with information provided by the model checker output. In addition, this work demonstrated the advantage of model checking over simulation in that simulators require the creation of "test input vectors." This is similar to the drawback associated with testing. In both testing and simulation, it is necessary to guess at where errors might occur since it is impractical to test all cases.

What is also significant in this work is the demonstration that the error could have been detected earlier in the design cycle—before the design was implemented in silicon, by applying model checking on a simplified specification. In this simplified specification and checking effort, the failure and associated causal information were provided by checking against ex-

pected "good" behavior and using counter example output information to understand the details of the failure.

4.2.5 Fujitsu Communications Chip

During field tests of a complex high speed communications IC chip, errors in the form of duplicated and lost data were observed shortly after powering on [Fujita 96]. The chip was designed for high speed switching operations at 156 MHz and involved over 110,000 gates. During the development effort, extensive simulation was used to validate the circuit. But since these errors occurred only after several seconds of operation hundreds of millions of simulation cycles would be required to fully simulate the error situation. Using simulation to investigate these errors was impractical.

To evaluate the problem the Symbolic Model Verifier (SMV) model checker was used. In creating the model the complex system was substantially abstracted and data path widths were reduced to lower the number of possible states of the model. Using this model in SMV running on a workstation, it required less than a half-hour to determine the errors arose from a condition where the same address appeared twice in a FIFO.

In this case, model checking was able to identify an error that eluded detection throughout the design and conventional verification processes. The complexity of the model used was substantially less than the chip itself, providing evidence that a simplified model of a complex system can be effective in identifying errors that have significant impact and are difficult to detect using conventional review and simulation techniques.

4.2.6 Software Model Checking

Wing and Vaziri-Farahani applied model-checking techniques to software [Wing 96]. Their approach abstracted the software into finite state models that are specific both to the system and the particular property or set of properties to be verified.

Two real-world cache coherence protocols were verified using their methods, for versions of the Andrew File System [Howard 88], and for a modified version of the Coda Distributed File System [Mummert 94]. The SMV model checker was used for the analysis. For the most complex protocol model, SMV took less than 1 second to check over 43,600 reachable states.

What is significant about this investigation is that it applied a hardware model-checking tool to a software system and highlighted the importance of engineering judgment in choosing the right abstractions. Their abstraction choices were based upon their intuition. But while also noting that there is a need to research more formal techniques to make the reasoning about abstraction choices more precise, the "'approximate' reasoning by the use of model checking is a low-cost, yet highly effective" approach to finding errors in software system designs.

4.3 Advantages of Model-Checking and MBV Techniques

There is substantial evidence within the successes of the use of model checking that errors can be detected early and that these errors can be identified using relatively simple models of highly complex systems [Clarke 95, Dill 92, Raimi 97]. In addition, it has been shown that many of the errors uncovered by modeling checking techniques are subtle and often highly problematic errors. These subtle errors eluded detection through conventional reviews, simulation, and testing.

Collectively, these results suggest a potential for the successful application of model-based verification to software intensive systems. Specifically, by using simplified and focused models of complex software systems, model checking offers the potential to augment existing software defect identification practices (e.g., inspections, simulation, testing) and aid in identifying problematic errors that elude detection with these practices.

4.4 Model-Checking Tools

Commercial model-checking tools for digital hardware system design are now available. They are generally packaged as part of a more comprehensive development environment for microprocessor or custom integrated chip design and verification (FormalCheck, Insight). Other model-checking tools have been used within commercial organizations for both hardware and communications protocols (SPIN). Many more model-checking tools are available at academic and research institutions (e.g., SMV, Murphi, Verdict). A representative set of these tools is presented in Appendix A.

5 Summary

Model-based verification is an amalgamation of a variety of modeling and formal analysis techniques that are either already established software engineering practices, being used in other engineering domains (e.g., commercial microprocessor design), or emerging in academic and corporate research communities. The technical foundations of model-based verification establish a basis for improving the dependability of software upgrades by addressing upgrades at higher design levels.

Through their focus on abstract essential models of software systems, model-based verification technologies offer the potential to detect errors early in the life cycle. Other benefits of these technologies include:

- provision of an infrastructure for design and upgrade rationale
- establishment of a systematic basis for integrated verification and testing practices
- facilitation of an evolutionary upgrade paradigm based upon pragmatic formal modeling methodologies

The objectives of the Verification and Test efforts within the SEI are to investigate and codify technologies for improving the engineering practices and dependability of software system upgrades. While there are pragmatic issues that must be addressed, model-based verification technologies (e.g., model checking) are sufficiently matured that they can be documented, structured into a defined practice, and applied to real world problems in the form of pilot software engineering studies. By focusing on the most mature and promising technical approaches and applying those techniques to real world problems, these studies will help to determine the cost and technical effectiveness of model-based verification practices, and establish a foundation for broader implementation in DoD programs. Most of these studies will be collaborations with organizations interested in adopting the practices. Continuing cooperative investigations with research and educational institutions are being conducted to ensure mathematical soundness, clarify related technical issues, and understand implementation constraints.

The results to date of the work in model-based verification suggest that model-based verification approaches can become complementary components of a comprehensive verification practice for software upgrades. As such, they can be integrated into and augment current software upgrade practices by providing additional techniques for error detection and correction. Future SEI technical reports and publications will present the results of the ongoing work and describe model-based verification practices in greater detail.

References

[Ackoff 62] Ackoff, R.L. *Scientific Method: Optimizing Applied Research Decisions*. New York: Wiley, 1962.

[Atlee 93] Atlee, J. & Gannon, J. "State-Based Modeling Checking of Event-Driven System Requirements," *IEEE Transactions on Software Engineering 19*, 1 (January 1993): 24-40.

[Clarke 94] Clarke, E. M.; Grumberg, O.; & Long, D. "Model Checking and Abstraction," *ACM Transactions on Programming Languages 16*, 5 (September 1994): 1512-1542.

[Clarke 95] Clarke, Edmund M., et al. "Verification of the Futurebus+ Cache Coherence Protocol." *Formal Methods in System Design 6* (1995): 217-232.

[Clarke 96a] Clarke, E. & Kurshan, R. "Computer Aided Verification." *IEEE Spectrum 33*, 6 (1996): 61-67.

[Clarke 96b] Clarke, E. M. & Wing, Jeannette. "Formal Methods: State of the Art and Future Directions." *ACM Computing Surveys 28*, 4 (December 1996): 626-643.

[Clements 95] Clements, P. *Formal Methods in Describing Architectures* [online]. <http://www.sei.cmu.edu/publications/articles/formal.methods.desc.architectures.html>, Wednesday, September 3, 1998; 2:29 P.M. EDT.

[Computer 96] *IEEE Computer 29*, 4 (April 1996): 16-30.

[Craigen 93a] Craigen, D.; Gerhart, S.; & Ralston, T. *An International Survey of Industrial Applications of Formal Methods, Volume I: Purpose, Approach, Analysis and Conclusions*. Gaithersburg, Md.: U. S. Department of Commerce, Technology Administration, National Institute of Standards and Technology. Computer Systems Laboratory (March 1993).

[Craigen 93b] Craigen, D.; Gerhart, S.; & Ralston, T. *An International Survey of Industrial Applications of Formal Methods, Volume II: Case Studies*. Gaithersburg, Md.: U. S. Department of Commerce, Technology Administration, National Institute of Standards and Technology. Computer Systems Laboratory (March 1993).

[De Palma 96] De Palma, Gary, & Glasser, Arthur. "Formal Verification Augments Simulation." *Electronic Engineering Times* n885 (1996): 56

[Dill 92] Dill, D. L.; Drexler, A. J.; Hu, A. J.; & Yang, C. H. "Protocol Verification as a Hardware Design Aid," 522-525. *IEEE International Conference on Computer Design: VLSI in Computers and Processors*. Los Alimitos, Calif.:IEEE Computer Society Press, 1992.

[Easterbrook 97] Easterbrook, Steve & Callahan, John, "Formal Methods for V&V of Partial Specifications: An Experience Report." *Proceedings of the Third International Symposium on Requirements Engineering*, Annapolis, Md., January 5-8, 1997

[Easterbrook 98] Easterbrook, Steve & Callahan, John. "Formal Methods for V&V of Partial Specifications: A Case Study." *Journal of Systems and Software 40*, 3 (March 1998): 199- 210.

[Fujita 96] Fujita, M. "Debugging a Communications Chip." *IEEE Spectrum 33*, 6 (June 1996): 64.

[Gaudel 96] Gaudel, Marie-Claude & Woodcock (eds.) "FME '96: Industrial Benefit and Advances in Formal Methods, Lecture Notes in Computer Science 1051." *Proceedings of the Third International Symposium in Formal Methods Europe*. Oxford, UK, March 1996.

[Harel 87] Harel, David. *Algorithmics: The Spirit of Computing*. New York: Addison-Wesley, 1987.

[Hetzel 88] Hetzel, B. *The Complete Guide To Software Testing, 2nd Edition*. Wellesley, Mass.: QED Information Sciences, Inc., 1988.

[Heimdahl 96] Heimdahl, M. P. E. & Leveson, N. G. "Completeness and Consistency in Hierarchical State-Based Requirements." *IEEE Transactions on Software Engineering 22*, 6 (June 1996): 363-377.

[Heitmeyer 96] Heitmeyer, C.; Jeffords, R. D.; & Labaw, B. "Automated Consistency Checking of Requirements Specifications." *ACM Transactions on Software Engineering and Methodology 5*, 3 (July 1996): 231-261.

[Heitmeyer 97] Heitmeyer, C. & Bharadwaj, R. *Model Checking Complete Requirements Specifications Using Abstraction.* (NRL/MR/5540—97-7999). Center for High Assurance Systems, Naval Research Laboratory Memorandum Report, November 10, 1997.

[Howard 88] Howard, J.; Kazar, M.; Menees, S.; Nichols, D.; Satyanarayanan, M.; Sidebotham, R.: & West, M. "Scale and Performance in a Distributed File System." *ACM Transaction on Computing Systems 6*, 1 (1988): 51-81.

[Jackson 95] Jackson, M. *Software Requirements and Specifications: A Lexicon of Practice, Principles, and Prejudices.* New York: ACM Press, Addison-Wesley, 1995.

[Jackson 96a] Jackson, Daniel. "Requirements Need Form, Maybe Formality." *IEEE Computer 29*, 3 (March 1996): 21-22.

[Jackson 96b] Jackson, Daniel & Wing, Jeannette. "Lightweight Formal Methods." *IEEE Computer 29*, 4 (April 1996): 21-22.

[Jackson 96c] Jackson, Daniel, "Nitpick: A Checkable Specification Language." *Proceedings of Workshop on Formal Methods in Software Practice*, San Diego, Ca., January 10-11, 1996. New York: ACM Press, 1996.

[Jackson 96d] Jackson, Daniel & Damon, Craig. *Nitpick Reference Manual.* Pittsburgh, Pa.: Carnegie Mellon University, 1996.

[Jackson 98] Jackson, M. "Formal Methods and Traditional Engineering." *Journal of Systems and Software*, New York: Elsevier Science, Inc. March 1998: 191-194.

[Klein 93] Klein, M.; et al. *A Practitioner's Handbook for Real-Time Analysis: Guide to Rate Monotonic Analysis for Real-time Systems.* Boston, MA: Kluwer Academic Publishers, 1993.

[Leveson 94] Leveson, N.G.; Heimdahl, M. P. E.; Hildreth, H.; & Reese, J. D. "Requirements Specification for Process-Control Systems." *IEEE Transactions on Software Engineering 20*, 9 (September 1994): 684-707.

[Linger 94] Linger, R. C. "Cleanroom Process Model." *IEEE Software 11*, 2 (March 1994): 50-58.

[McMillan 92a] McMillan, Kenneth L. *The SMV System (DRAFT)*. [online] Available FTP: <ftp://emc.cs.cmu.edu/pub/smvmanual.r2.2.ps.Z> (1992).

[McMillan 92b] McMillan, Kenneth L. *Symbolic Model Checking: An Approach to the State Explosion Problem*. (CMU-CS-92-131, ADA250924). Pittsburgh, Pa.: Carnegie Mellon University, 1992.

[McMillan 93] McMillan, Kenneth L. *Symbolic Model Checking*. Norwell, Mass: Kluwer Academic Publishers, 1993.

[Meyer 85] Meyer, B. "On Formalism in Specification." *IEEE Software 2*, 1 (January 1985): 6-26.

[Miller 98] Miller, S. P. "Specifying the Mode Logic of a Flight Guidance System in CoRE and SCR," pp. 44-53. *Proceedings of Second Workshop on Formal Methods in Software Practice*. Clearwater Beach, Florida, March 4-5, 1998.

[Mills 87] Mills, H.; Dyer, M.; & Linger, R. "Cleanroom Software Engineering," *IEEE Software 4*, 5 (1987).

[Mummert 94] Mummert, L. & Satyanarayanan, M. "Large Granularity Cache Coherence for Intermittent Connectivity," [summary] 279-289. *Proceedings of the USENIX Summer Conference*. Boston, Mass., June 1994. Berkeley, Calif.: USENIX Association.

[NASA 95] National Aeronautics and Space Administration. *Formal Methods for Specification and Verification Guidebook for Software and Computer Systems, Volume 1: Planning and Technology Insertion, Office of Safety and Mission Assurance*. Washington, D.C.: July 1995.

[NASA 97] National Aeronautics and Space Administration. *Formal Methods for Specification and Verification Guidebook for Software and Computer Systems, Volume 2: A Practitioner's Companion*. Washington, D.C.: May 1997.

[Nitpick 98] Jackson, Daniel & Damon, Craig. <http://www.cs.cmu.edu/~nitpick/>, Wednesday, September 3, 1998; 11:31 A.M. EDT.

[Parnas 92] Parnas, D. L. *Tabular Representations of Relations, CRL Report 260*. TRIO (Telecommunications Research Institute of Ontario), Hamilton, Ontario, Canada: McMaster University, October, 1992.

[Parnas 98] Parnas, D. L. "'Formal Methods' Technology Transfer Will Fail." *Journal of Systems and Software 40*, 3. (March 1998): 195-198.

[Place 89] Place, P. R. H. & Wood, W. "Formal Specification Methods in Practice." *Software Engineering Institute Annual Technical Review* (January 1989): 1-9. Sponsored by the Department of Defense. Grant/Contract No. F19628-25-C-0003.

[Place 90] Place, P. R. H.; Wood, W.; & Tudball, M. *Survey of Formal Specification Techniques for Reactive Systems* (CMU/SEI-90-TR-5, ADA223741). Pittsburgh, Pa.: Software Engineering Institute, Carnegie Mellon University, May 1990.

[Place 93] Place, P. R. H., & Kang, K. C. *Safety-Critical Software: Status Report and Annotated Bibliography* (CMU/SEI-93-TR-005, ADA266993). Pittsburgh, Pa.: Software Engineering Institute, Carnegie Mellon University, June 1993.

[Pfleeger 97] Pfleeger, S. L. & Hatton, L. "Investigating the Influence of Formal Methods," *IEEE Computer 30*, 2. (February 1997): 33-43.

[Raimi 97] Raimi, R. & Lear, J. "Analyzing a PowerPC™ 620 Microprocessor Silicon Failure Using Model Checking," 964-973. *Proceedings of the International Test Conference 1997*, Washington, D.C., November 1-6, 1997.

[Rumbaugh 91] Rumbaugh, J; Blaha, M.; Premerlani, W; Eddy, F; & Lorensen, W. *Object-Oriented Modeling and Design*, , Englewood Cliffs, New Jersey: Prentice Hall, 1991.

[Rushby 93] Rushby, J. *Formal Methods and the Certification of Critical Systems* (SRI-CSL-93-07). Menlo Park, Calif.: SRI International, November 1993.

[SMV 98] Beregey, Serezin. "Formal Methods – Model Checking." <http://www.cs.cmu.edu/~modelcheck/>, Wednesday, September 3, 1998; 11:32 A.M.

[USAF 96] Department of the Air Force, Software Technology Support Center. *Guidelines for Successful Acquisition and Management of Software-Intensive Systems: Weapon Systems, Command and Control Systems, Management Information Systems, Volume 1*. Washington, DC: June 1996.

[Vienneau 93] Vienneau, Robert. *A Review of Formal Methods* (RL/C3C). Griffins AFB, N.Y.: Rome Laboratory, May 26, 1993.

[Wing 90] Wing, J. "A Specifier's Introduction to Formal Methods," *IEEE Computer 23*, 9 (September 1990): 8-22.

[Wing 96] Wing, Jeannette & Vaziri-Farahani, Mandana. *A Case Study in Model Checking Software Systems* (CMU-CS-96-124, ADA309147). Pittsburgh, Pa.: Carnegie Mellon University, 1996.

Appendix A: Model Checking and Related Verification Tools and Techniques

There are a number of model-checking tools that have been developed and successfully employed to analyze systems. This section presents a summary of a representative set of those tools.

Generally, a tool reflects a particular theoretical approach. Individual tools are often distinguished from others that use a similar theoretical approach by the specific search or analysis algorithms implemented in the tool or a unique implementation of the approach.

A.1 Nitpick

Nitpick is a tool for checking software requirements and design specifications [Jackson 96c, Jackson 96d, Nitpick 98]. A state machine model of the system is created using a Z-like language (NP). Within the Nitpick tool the NP model is checked against expected properties (claims) of the system. These properties of the system as well as the model are expressed in the NP language.

The Nitpick tool compares the expected properties of the system (claims) with the model and confirms that the claim is true or, if it not true, provides a counter example that demonstrates the violation of the expected property. This is shown in Figure A1. The input file (specification) is a text file that consists of both the model and claims expressed in the NP language.

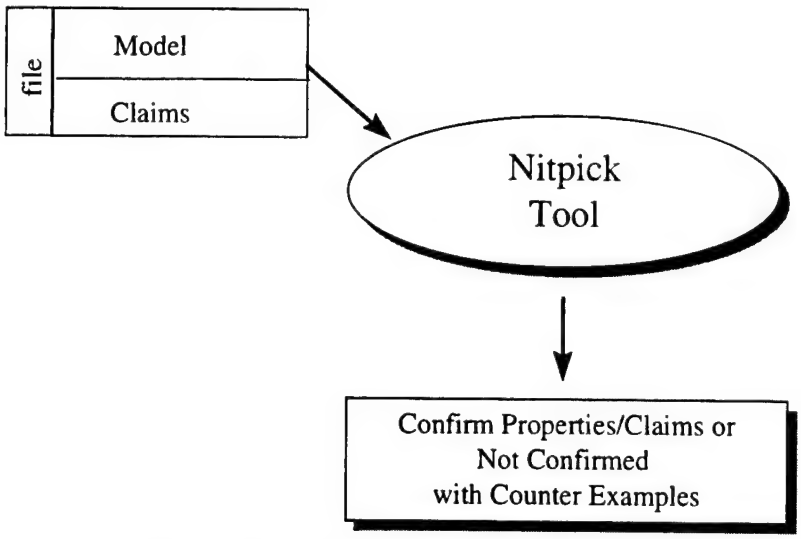


Figure A1: The Nitpick Tool

More information on Nitpick can be found at the Web site: http://www.cs.cmu.edu/~nitpick/.

A.2 Symbolic Model Verifier (SMV)

The Symbolic Model Verifier (SMV) model checker is a tool for checking a finite state model of a system against properties, assertions or claims (a specification) of that system. Models of the system are expressed in the SMV input language [McMillan 92a, McMillan 92b, McMillan 93, SMV 98]. The desired properties (specifications) of the system are expressed in computational tree logic (CTL).

A system model may include multiple state machine models and associated properties or claims about the system. The models and associated properties are grouped into modules. The sets of modules for a system are input into the SMV model checker system as shown in Figure A2.

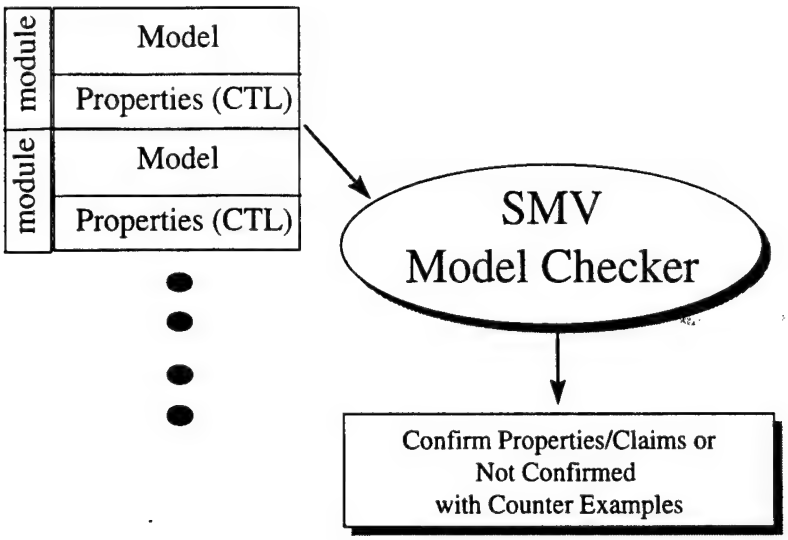


Figure A2: SMV Model Checker

More information on SMV and model checking can be found at the Web site:
http://www.cs.cmu.edu/~modelcheck/

A.3 Software Cost Reduction (SCR)

The Software Cost Reduction (SCR) requirements method is a system and software requirements specification approach for real-time embedded systems. The SCR approach is based on formal methodologies and is capable of specifying both functional and non-functional (e.g., timing and accuracy) requirements.

This is not strictly a model-checking tool but is a tabular specification technique that incorporates formal methodologies for describing state machines. As such it can be employed as a modeling approach for abstract representations of a system. In addition, the tool interfaces to other model-checking tools (e.g., SPIN, SMV) to enable more extensive model checking.

The SCRtool can relieve a reviewer of mundane, application-independent activities. For example, checks like ensuring that all variables are initialized, all variables are defined, etc., i.e. that the specification has all the "right" qualities required of any specification can be automated.

More information on the SCR tool can be found at the Web site:
http://www.itd.nrl.navy.mil/ITD/5540/personnel/heitmeyer.html

A.4 SPIN

SPIN is a model checker developed at Bell Labs for the analysis of synchronous concurrent systems (e.g., data communication protocols, distributed operating systems, database systems). Using a model specified in its custom language, Promela, SPIN can perform random simulations of a system's execution and/or generate a C program that, when compiled and executed, can be used to check a system's state space against expected properties.

For smaller problems, SPIN's model checking exhaustively explores every possible state of the system model. For larger problems, SPIN uses a hashing function to get best coverage of the state space, given the physical limitation of the computing environment. Specifically, SPIN uses partial order reduction to reduce the number of states that must be explored in order to verify a system and in the cases that the state space exceeds the computer's resources, SPIN employs bit-state hashing to selectively explore the state space.

Correctness properties are expressed in terms of Linear Temporal Logic, but these must first be translated into the Promela language for input into the tool. These can be translated manually or translated using the automated translator.

Additional information on SPIN can be found on the Web site:
http://netlib.bell-labs.com/netlib/spin/whatispin.html

A.5 Murphi

Murphi (actually Murϕ) is a high-level description language and associated tool set that enables automatic formal verification of protocols and other distributed finite-state systems [Dill 92]. The Murphi approach to verifying large systems is to downscale them. Instead of testing a small fraction of the possibilities for a large model of the system, Murphi assesses all of the possibilities for a small instance of the system.

The Murphi description language supports nondeterministic, scalable descriptions and employs a set of iterated guarded commands. Murphi can detect deadlocks (states with no successors other than themselves). Also, rules can give error conditionals and/or assertions. Finally, if any of the invariants are violated, an error is reported. The system cannot detect livelocks.

Additional information on Murphi can be found at the Web site:
http://sprout.stanford.edu/dill/murphi.html

A.6 Commercial Model-Checking and Related Products

A number of commercial model-checking tools, primarily for digital hardware systems, are available. They are generally packaged as part of a more comprehensive development environment for microprocessor or custom integrated chip design and verification. Other tools

have been used within commercial organizations for both hardware and communications protocols. Example products that are commercially available or are in house development tools include:

- FormalCheck. FormalCheck is a model checker developed by Lucent Technologies that is used in the design and verification of digital hardware systems. In applying FormalCheck a user inputs the design model in VHDL or Verilog RTL code and provides a query that describes a desired property of the design or a set of constraints on primary inputs of the design. The constraints limit the set (or sequence) of possible values on those inputs consistent with the legal operation of the design. As with other model checkers, FormalCheck either verifies that the property is true under all possible, allowable conditions, or provides an error trace showing a case where the property fails. FormalCheck can be used for communication protocols, bus controllers, video applications, control portions of processors and cache protocols. A Web site for more information is http://www.bell-labs.com/org/blda/product_formal.html.
- Insight. Chrysalis Symbolic Design, Inc. offers software tools that use formal methods to automate the design of digital integrated circuits. These verification tools include the Design INSIGHT® Assertions™ formal model-checking tool which provides a generalized set of programmable formal checks that engineers can use to eliminate typical problems, such as drive fights and one-hot controller protocols as they design and debug intra-chip interfaces and protocol logic." A Web site for more information is http://www.chrysalis.com.
- SpecTRM (RSML). The tool set, Specification Tools and Requirements Methodology (SpecTRM) is used for system requirements specification. Its formal requirements specification language is SpecTRM-RL, the successor language to Requirements State Machine Language (RSML) [Leveson 94, Heimdahl 96]. While the language is readable by engineers not expert in formal methods, it is amenable to automated analysis. The automated analysis enables early hazard and error identification and is especially useful for high assurance and safety systems. This tool set is representative of the approach of bring formal methods into software engineering practice by making the rather esoteric formal methodologies easier to use. Additional information on SpecTRM can be found at the Web site: http://www.safeware-eng.com/.
- Verdict. Verdict is a model checker that was developed within Motorola. It is a computational tree logic-based model checker that includes facilities for parsing the IBM hardware description language DSL™. It has been used for digital hardware analysis at the joint IBM-Motorola Somerset design center [Raimi 97].

REPORT DOCUMENTATION PAGE

Form Approved
OMB No. 0704-0188

1. AGENCY USE ONLY (LEAVE BLANK)	2. REPORT DATE	3. REPORT TYPE AND DATES COVERED
	September 1998	Final

4. TITLE AND SUBTITLE
Model-Based Verification: A Technology for Dependable System Upgrade

5. FUNDING NUMBERS
C — F19628-95-C-0003

6. AUTHOR(S)
David P. Gluch & Charles B. Weinstock

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)
Software Engineering Institute
Carnegie Mellon University
Pittsburgh, PA 15213

8. PERFORMING ORGANIZATION REPORT NUMBER
CMU/SEI-98-TR-009

9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)
HQ ESC/DIB
5 Eglin Street
Hanscom AFB, MA 01731-2116

10. SPONSORING/MONITORING AGENCY REPORT NUMBER
ESC-TR-98-009

11. SUPPLEMENTARY NOTES

12.A DISTRIBUTION/AVAILABILITY STATEMENT
Unclassified/Unlimited, DTIC, NTIS

12.B DISTRIBUTION CODE

13. ABSTRACT (MAXIMUM 200 WORDS)

This is a preliminary report on the technological foundations of model-based verification for engineering software system upgrades. It describes the historical background and technical foundations for the approach and begins to provide a basis for the transition of model-based verification into practice. Critical technical and procedural issues that have been or are being addressed to ensure successful transition are examined. The report is aimed at providing technical insight and understanding for software management and engineering personnel on this emerging technology for verification of software system upgrades.

14. SUBJECT TERMS
software modeling, model-checking technologies, software upgrades, software management

15. NUMBER OF PAGES
44

16. PRICE CODE

17. SECURITY CLASSIFICATION OF REPORT	18. SECURITY CLASSIFICATION OF THIS PAGE	19. SECURITY CLASSIFICATION OF ABSTRACT	20. LIMITATION OF ABSTRACT
UNCLASSIFIED	UNCLASSIFIED	UNCLASSIFIED	UL

NSN 7540-01-280-5500

Standard Form 298 (Rev. 2-89)
Prescribed by ANSI Std. Z39-18
298-102